A Game-Theoretic Approach to Coalition Formation in Green Cloud Federations

Marco Guazzone*, Cosimo Anglano†, Matteo Sereno*
*Department of Computer Science, University of Torino, Italy
†Department of Science and Technological Innovation, University of Piemonte Orientale, Italy

***Abstract*—Federations among sets of Cloud Providers (CPs), whereby a set of CPs agree to mutually use their own resources to run the VMs of other CPs, are considered a promising solution to the problem of reducing the energy cost. In this paper, we address the problem of federation formation for a set of CPs, whose solution is necessary to exploit the potential of cloud federations for the reduction of the energy bill. We devise an algorithm, based on cooperative game theory, that can be readily implemented in a distributed fashion, and that allows a set of CPs to cooperatively set up their federations in such a way that their individual profit is increased with respect to the case in which they work in isolation.We show that, by using our algorithm and the proposed CPs' utility function, they are able to self-organize into Nash-stable federations and, by means of iterated executions, to adapt themselves to environmental changes. Numerical results are presented to demonstrate the effectiveness of the proposed algorithm.**

Keywords*—*Cloud Federation, Cooperative Game Theory, Coalition Formation

I. INTRODUCTION

Many modern Internet services are implemented as cloud applications consisting of a set of *Virtual Machines* (VMs) that are allocated and run on a physical computing infrastructure managed by a virtualization platform (e.g., Xen [1], VMware [2], etc.). These infrastructure are typically owned by a *Cloud Provider* (CP) (e.g., Amazon AWS, Rackspace, Windows Azure, etc.), and are located into a (set of possibly distributed) data center(s).

One of the key issues that must be faced by a CP is the reduction of its energy cost, that represents a large fraction of the total cost of ownership for physical computing infrastructures [3]. This cost is mainly due to the consumption of the physical resources that must be switched on to run the workload.

To reduce energy consumption, two techniques are therefore possible for a CP: (a) to minimize the number of hosts that are switched on by maximizing the number of VMs allocated on each physical resource (using suitable resource management techniques [4], [5]), and (b) to use resources that consume less energy.

Cloud federations [6], whereby a set of CPs agree to mutually use their own resources to run the VMs of other CPs, are considered to be a promising solution for the reduction of energy costs [7] as they ease the application of both techniques.

As a matter of fact, while each individual CP is bound to its specific energy provider and to the physical infrastructure it owns, a set of federated CPs may enable the usage of more flexible energy management strategies that, by suitably relocating the workload towards CPs that pay less for the energy, or that have more energy-efficient resources, may reduce the energy bill for each one of them.

In this paper, we address the problem of federation formation for a set of CPs, and we devise an algorithm that allows these CPs to decide whether to federate or not on the basis of the profit they receive for doing so. In our approach, each CP pays for the energy consumed by each VM, whether it belongs to its own workload or to the one of another CP, but receives a payoff (computed as discussed later) for doing so.

The algorithm we propose is based on cooperative game theory [8], [9]. In particular, we rely on *hedonic games* (see [10] for their definition) whereby each CP bases its decision on its own preferences. Depending on the specific operational conditions of each CP (i.e., the resource requirements of its workload, its cost of energy, the energy consumption of its physical machines, and the revenue it obtains when running each VM on these machines), different federations (each one consisting of a subset of the CPs), or even no federation at all, may be formed by the involved CPs. We call *federation set* the set of distinct federations formed by a set of CPs.

The algorithm we propose computes the federation set that results in the highest profit that can be achieved by a set of autonomous and selfish CPs. This derives from the fact that this algorithm ensures that all the federations formed by groups of CPs are *stable*, that is CPs have no incentive to leave the federation once they decide to participate.

Unlike similar proposals (e.g., [11]), we adopt a distributed approach in which each CP autonomously and selfishly makes its own decisions, and the best solution emerges from these decisions without the need to synchronize them or to resort to a trusted third party, and such that no CP can benefit by moving from her/his federation to another (possibly empty) one. In this way, we avoid three drawbacks that affect existing proposals, namely (1) the difficulty of finding a third party that is trusted by all the CPs, (2) the need to suspend the operations of all the CPs when the federation set is being computed, and (3) the instability inside the federation due to single CP's movements.

The rest of this paper is organized as follows. In Section II, we describe the system under study, and provide some simple motivating example. In Section III, we present a cooperative game-theoretic model of the system under study, and show stability conditions and profit allocation strategies that provide the theoretical foundation for the distributed coalition formation algorithm, that is also presented in this section. In Section IV, we present results from an experimental evaluation to show the effectiveness of the proposed approach. In Section V, we provide a short overview of related works. Finally, conclusions and an outlook on future extensions are presented in Section VI.

II. Problem Formulation

We consider a set of N CPs denoted by $\mathcal{N} = \{1, 2, \dots, N\}$, where each CP i is endowed with a set $\mathcal{H}_i$ of physical hosts. We denote as $\mathcal{H} = \bigcup_{i=1}^{N} \mathcal{H}_i$ the set of all the hosts collectively belonging to the various CPs. These hosts are grouped into a set $\mathcal{G}$ of *host classes* according to their processor type and to the amount of physical memory they provide; all the hosts in the same class are homogeneous in terms of processor and memory size. For any $h \in \mathcal{H}$ we denote by $g(h)$ the function that gives the host class of h (i.e., a function $g : \mathcal{H} \rightarrow \mathcal{G}$).

As discussed in [12], we assume that a host h consumes $C_{g(h)}^{\min}$ Watts when its CPU is in the idle state, $C_{g(h)}^{\max}$ Watts when its CPU is fully utilized, and $\left(C_{g(h)}^{\min} + f \cdot \left(C_{g(h)}^{\max} - C_{g(h)}^{\min}\right)\right)$ when a fraction $f \in [0, 1]$ of its CPU capacity is used. This model, albeit simple, has been shown to provide accurate estimates of power consumption for different host types when running several benchmarks representative of real-world applications [12].

Physical hosts run cloud workloads, consisting in a set $\mathcal{J} = \bigcup_{i=1}^{N} \mathcal{J}_i$ of VMs, where $\mathcal{J}_i$ denotes the set of VMs that compose the workload of the i-th CP (each VM contains the whole execution environment of one or more applications).

As typically done by CPs, VMs are grouped into a set $\mathcal{Q}$ of *VM classes* according to the computing capacity provided by their virtual processors, and to the amount of physical memory they are equipped with; all the VMs belonging to the same class provide the same amount of computing capacity and of physical memory. For instance, Amazon's EC2 [13] defines the *Elastic Compute Unit (ECU)* as an abstract computing resource able to deliver a capacity equivalent to that of a 1.2 GHz 2007 Xeon processor, and provides various *instance types* (that are equivalent to our VM classes) that differ among them in the number of ECUs and in the amount of RAM they are equipped with. More specifically, *small*, *medium*, and *large*, corresponding to VM class 1, 2, and 3, respectively, provide 1 ECU and 1.7 GB of RAM, 2 ECUs and 3.7 GB of RAM, and 4 ECUs and 7.5 GB of RAM, respectively.

For any VM $j \in \mathcal{J}$, we denote by $q(j)$ the function that gives its VM class (i.e., a function $q : \mathcal{J} \rightarrow \mathcal{Q}$). Using this notation, for any $j \in \mathcal{J}$ we denote by $CPU_{q(j)}$ and by $RAM_{q(j)}$ the amount of computing capacity and of physical memory of VM j, respectively. As an example, in the Amazon EC2 case we have that $CPU_1 = 1$ ECU and $RAM_1 = 1.7$ GB, while $CPU_3 = 4$ ECU and $RAM_3 = 7.5$ GB.

When allocated on a physical host h, a VM j uses a certain fraction $A_{q(j),g(h)}$ of CPU capacity and a certain fraction $M_{q(j),g(h)}$ of physical memory. $A_{q(j),g(h)}$ can be determined by measuring, with a suitable benchmark (e.g., GeekBench [14]), the computing capacity Cap_v delivered by the virtual processor of VMs in $q(j)$ and the capacity Cap_p delivered by the physical processor of hosts in $g(h)$, and then by dividing these quantities, i.e., $A_{q(j),g(h)} = \frac{Cap_v}{Cap_p}$. For instance, if $Cap_v = 1,000$ and $Cap_p = 8,000$, then $A_{q(j),g(h)} = 0.125$. $M_{q(j),g(h)}$ can instead be computed as $M_{q(j),g(h)} = \left\lceil \frac{RAM_{q(j)}}{\text{RAM size of hosts in } g(h)} \right\rceil$.

Each CP i charges, for each VM j, a *revenue rate* (that depends on the class $q(j)$ of that VM) that specifies the amount of money that the VM owner must correspond per unit of time. For instance, Amazon charges 0.08 \$/hour, 0.16 \$/hour, and 0.32 \$/hour for *small*, *medium*, and *large* instances, respectively. Consequently, CP i earns a global revenue rate that is given by the sum of the revenue rates of individual VMs. To run its workload, CP i incurs an energy cost quantified by the *energy cost rate* (the amount of money that is paid per unit of time), which is the energy cost resulting from the allocation of (a subset of) the workload $\mathcal{J}$ on its host set $\mathcal{H}_i$ that must be paid per unit of time (see Section III-C for a discussion on the optimization technique we use to minimize it). We define the *net profit rate* of CP i as the difference between its global revenue rate (obtained for hosting a set of VMs) and its global energy cost rate (that it incurs to run such VMs).

Our goal is to allocate all the VMs in $\mathcal{J}$ on the hosts in $\mathcal{H}$ (independently from the corresponding CPs) in such a way to maximize the *net profit rate* of each CP i. This goal can be achieved by finding the smallest set of hosts that are sufficient to accommodate the resource shares of all the VMs in $\mathcal{J}$ such that the overall energy consumption is minimized, and by providing a suitable revenue for those CPs that host VMs belonging to other CPs.

III. The Cooperative CP Game

To cooperate, a set of CPs must first form a *coalition*, i.e., they all must agree to share their own resources and users among them. Given a set of CPs, however, there are many different coalitions that can be formed, each one differing from the other ones in terms of the profit they bring to their members. Therefore, a way must be provided to each CP to decide which coalition to join. A CP must indeed consider various factors before deciding whether to join or not a coalition, the main ones being:

- *Stability*: a coalition is *stable* if none of its participants finds that it is more profitable to leave it (e.g., to stay alone or to join another federation) rather than cooperating with the other ones. Lack of stability causes possible monetary losses for the following reasons: *i)* a CP that has joined a coalition with the expectation of receiving users from other CPs is penalized if, after switching on additional resources to accommodate these users, these other CPs leave the coalition; *ii)* a CP that has accepted more users than those that it can serve without incurring into a penalty, expecting to use the resources of other CPs to accommodate them, is penalized if these CPs leave the coalition.

- *Fairness*: when joining a federation, a CP expects that the resulting profits are fairly divided among participants. As unfair division leads to instability, it is necessary to design an allocation method that ensures fairness.

In this section, we model the problem of coalition formation as a *coalition formation cooperative game with transferable utility* [8], [9].

A. Characterization

Our coalition formation algorithm is based on a *hedonic game* [10], [15], a class of *coalition formation cooperative games* [9], [15], [16] where each player acts as a selfish agent and where her/his preferences over coalitions depend only on the composition of his coalition.

Given the set $\mathcal{N} = \{1, 2, \ldots, N\}$ of CPs (henceforth also referred to as the *players*), a *coalition* $\mathcal{S} \subseteq \mathcal{N}$ represents an agreement among the CPs in $\mathcal{S}$ to act as a single entity.

At any given time, the set of players is partitioned into a *coalition partition* Π, that we define as the set $\Pi = \{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_l\}$. That is, for $k = 1, \ldots, l$, each $\mathcal{S}_k \subseteq \mathcal{N}$ is a disjoint coalition such that $\bigcup_{k=1}^{l} \mathcal{S}_k = \mathcal{N}$ and $\mathcal{S}_j \cap \mathcal{S}_k = \emptyset$ for $j \neq k$. Given a coalition partition Π, for any CP $i \in \mathcal{N}$, we denote by $\mathcal{S}_\Pi(i)$ the coalition $\mathcal{S}_k \in \Pi$ such that $i \in \mathcal{S}_k$.

Each coalition $\mathcal{S}$ is associated with its *coalition value* $v(\mathcal{S})$, that we define as the net profit of coalition $\mathcal{S}$, that is:

$$v(\mathcal{S}) = r(\mathcal{J}_\mathcal{S}) - e(\mathcal{J}_\mathcal{S}, \mathcal{H}_\mathcal{S}) \tag{1}$$

where:

- $r(\mathcal{J}_\mathcal{S})$ is the *revenue rate* of the coalition, and is given by the sum of revenue rates of individual VMs $j \in \mathcal{J}_\mathcal{S}$;
- $e(\mathcal{J}_\mathcal{S}, \mathcal{H}_\mathcal{S})$ is the *energy cost rate*, and can be derived by minimizing the energy cost resulting from the allocation of the workload $\mathcal{J}_\mathcal{S}$ on the host set $\mathcal{H}_\mathcal{S}$ (we discuss this in Section III-C).

Obviously, each CP $i \in \mathcal{S}$ must receive a fraction $x_i(\mathcal{S})$ of the coalition value, that we call the *payoff* of i in $\mathcal{S}$. Our game is conceived in such a way to form coalitions in which CPs get payoffs as high as possible, without violating the fairness requirement, so that stability is achieved. Thus, a *payoff allocation rule* must be specified in order to compute the payoffs of each coalition member and to ensure fairness in the division of payoffs.

To this end, we use the *Shapley value* [17], a payoff allocation rule that is based on the concept of players' *marginal contribution* (i.e., the change in the worth of a coalition when a player joins to that coalition), such that the larger is the contribution provided by a player to a coalition, the higher is the payoff allocated to it. [1] This means that, in a given coalition, some "more-contributing" CPs will be rewarded by other "less-contributing" CPs to encourage them to join the coalition. More specifically, the Shapley value $\phi_i(v)$ of player i is defined as:

$$\phi_i(v) = \sum_{\mathcal{S} \subseteq \mathcal{N} \setminus \{i\}} \frac{|\mathcal{S}|!\big(N - |\mathcal{S}| - 1\big)!}{N!} \Big(v\big(\mathcal{S} \cup \{i\}\big) - v(\mathcal{S}) \Big) \tag{2}$$

where the sum is over all subsets $\mathcal{S}$ not containing i. [2]

[1] More specifically, we use the *Aumann-Dréze* value [18], which is an extension of the Shapley value for games with coalition structures.

[2] The symbol "\" denotes the set difference operator, and the symbol "!" denotes the factorial function.

To set up the coalition formation process, we need to define a *preference relation* so that each CP can order and compare all the possible coalitions it belongs to, and hence it can build preferences over them. Formally, for any CP $i \in \mathcal{N}$, a *preference relation* $\succeq_i$ is defined as a complete, reflexive, and transitive binary relation over the set of all coalitions that CP i can form (see [10]). Specifically, for any CP $i \in \mathcal{N}$ and given $\mathcal{S}_1, \mathcal{S}_2 \subseteq \mathcal{N}$, the notation $\mathcal{S}_1 \succeq_i \mathcal{S}_2$ means that CP i prefers being a member of $\mathcal{S}_1$ over $\mathcal{S}_2$ or at least i prefers both coalitions equally. The strict counterpart of $\succeq_i$ is denoted by $\succ_i$ and implies that i strictly prefers being a member of $\mathcal{S}_1$ over $\mathcal{S}_2$. Note that the definition of a preference relation is one of the peculiarities of the coalition formation process, and, in general, it can be a function of several parameters.

In our coalition formation CP game, for any CP $i \in \mathcal{N}$, we use the following preference relation:

$$\mathcal{S}_1 \succeq_i \mathcal{S}_2 \Longleftrightarrow f_i(\mathcal{S}_1) \geq f_i(\mathcal{S}_2) \tag{3}$$

where $\mathcal{S}_1, \mathcal{S}_2 \subseteq \mathcal{N}$ are two coalitions containing CP i, and $f_i(\cdot)$ is a preference function, defined for any CP $i \in \mathcal{N}$ and any coalition $\mathcal{S}$ containing i, such that:

$$f_i(\mathcal{S}) = \begin{cases} x_i(\mathcal{S}), & \text{if } \mathcal{S} \notin h(i), \\ -\infty, & \text{otherwise.} \end{cases} \tag{4}$$

where $x_i(\mathcal{S})$ is the payoff received by CP i in $\mathcal{S}$, and $h(i)$ is a history set where CP i stores the identity of the coalition that it visited and left in the past. The rationale behind the use of $h(\cdot)$ is to avoid that a CP visit the same set of coalitions twice (a similar idea has also been used in previously published work, such as in [19], [20]). Thus, according to Eq. (4), each CP prefers to join to the coalition that provides the larger payoff, unless it has already been visited and left in the past. The strictly counterpart $\succ_i$ of $\succeq_i$ is defined by replacing $\geq$ with $>$ in Eq. (3).

B. The Coalition Formation Algorithm

The algorithm we propose is based on the following *hedonic shift rule* (see [21]): given a coalition partition $\Pi = \{\mathcal{S}_1, \ldots, \mathcal{S}_l\}$ on the set $\mathcal{N}$ and a preference relation $\succ_i$, any CP $i \in \mathcal{N}$ decides to leave its current coalition $\mathcal{S}_\Pi(i)$ to join another one $\mathcal{S}_k \in \Pi \cup \emptyset$ if and only if $\mathcal{S}_k \cup \{i\} \succ_i \mathcal{S}_\Pi(i)$, that is if its payoff in the new coalition exceeds the one it is getting in its current coalition. This shift rule can be seen as a selfish decision made by a CP to move from its current coalition to a new one, *regardless of the effects of this move on the other CPs.*

More precisely, starting from the initial setting where there are no coalitions and the history set of each CP is empty (i.e., $\Pi_0 = \big\{\{1\}, \{2\}, \ldots, \{N\}\big\}$ and $h(i) = \emptyset$, for all $i \in \mathcal{N}$), whenever it deems it is appropriate, each CP executes the actions listed in Figure 1 asynchronously with respect to the other CPs.

In Step 1, CP i evaluates all the possible coalitions it can form, starting by leaving its current one $\mathcal{S}_{\Pi_c}(i)$ to join another already existing coalition $\mathcal{S}_k$ by applying the corresponding hedonic shift rule. If such a coalition is found, in Step 2 CP i adds to its history set $h(i)$ the coalition $\mathcal{S}_{\Pi_c}(i)$ it is leaving, and updates the partition set by updating both $\mathcal{S}_k$ (that

```
Step 1: Coalition Formation Stage I. Given the current coalition partition $\Pi_c = \{\mathcal{S}_1, \ldots, \mathcal{S}_l\}$, each CP $i$ evaluates possible hedonic shift operations, in order to find a coalition $\mathcal{S}_k \in \Pi_c \cup \emptyset$ (if any) such that:
            $\mathcal{S}_k \cup \{i\} \succ_i \mathcal{S}_{\Pi_c}(i)$.
        Specifically, for each $\mathcal{S}_k \in \Pi_c \cup \emptyset$:
            1) Build $\mathcal{S} = \mathcal{S}_k \cup \{i\}$
            2) Retrieve the coalition values $v(\mathcal{S})$ and $v(\mathcal{S}_{\Pi_c}(i))$ (see Eq. (1))
            3) Retrieve the payoff values $x_i(\mathcal{S})$ and $x_i(\mathcal{S}_{\Pi_c}(i))$ (see Eq. (2))
            4) Compute the preference function values $f_i(\mathcal{S})$ and $f_i(\mathcal{S}_{\Pi_c}(i))$ (see Eq. (4))
            5) Evaluate the preference of $\mathcal{S}$ with respect to $\mathcal{S}_{\Pi_c}(i)$ as in Eq. (3)
Step 2: Coalition Formation Stage II. If such coalition $\mathcal{S}_k$ is found, CP $i$ decides to perform the hedonic shift rule to move to $\mathcal{S}_k$:
            1) CP $i$ updates its history $h(i)$ by adding $\mathcal{S}_{\Pi_c}(i)$.
            2) CP $i$ leaves its current coalition $\mathcal{S}_{\Pi_c}(i)$ and joins the new coalition $\mathcal{S}_k$.
            3) $\Pi_c$ is updated:
        $\Pi_{c+1} = \Big(\Pi_c \backslash \big\{\mathcal{S}_{\Pi_c}(i), \mathcal{S}_k\big\}\Big) \cup \Big\{\mathcal{S}_{\Pi_c}(i) \backslash \{i\}, \mathcal{S}_k \cup \{i\}\Big\}$.
        Otherwise, CP $i$ remains in the same coalition so that:
            $\Pi_{c+1} = \Pi_c$
```

Fig. 1. The Coalition Formation Algorithm

now contains also i) and $\mathcal{S}_{\Pi_c}(i)$ (that now does not contain i anymore).

The algorithm outlined can be readily implemented in a distributed fashion, given that each CP can act independently and asynchronously from any other CP in the system. It is however necessary to provide suitable mechanisms for:

- *state retrieval*: the algorithm assumes that each CP is able to obtain the current coalition partition Π_c. Any of the mechanisms of this type available in the literature (e.g., [22], [23]) can be used for this purpose;
- *atomic state update*: for the sake of correctness, Π_c must not change while CP i is performing Steps 1 and 2. Any of the distributed mutual exclusion algorithms available in the literature (e.g., [24], chap. 9) can be used for this purpose.

It is worth noting that the asynchronicity of our algorithm makes it suitable to be executed, for instance, when new users arrive to CPs, thus making it able to adapt to environmental changes. Our algorithm enjoys two important properties, that are proved below, namely:

1) it always converges to a stable partition, so the algorithm always terminate in a finite number of steps;
2) any final partition Π_f it finds is *Nash-stable*, that is no CP has incentive to move from its current coalition $\mathcal{S}_{\Pi_f}(i)$ to join a different coalition of Π_f, or to act alone. More formally (e.g., see [10]), a partition $\Pi = \{\mathcal{S}_1, \ldots, \mathcal{S}_l\}$ is *Nash-stable* if $\forall i \in \mathcal{N}$, $\mathcal{S}_\Pi(i) \succeq_i \mathcal{S}_k \cup \{i\}$ for all $\mathcal{S}_k \in \Pi \cup \emptyset$.

Proposition 1 (Convergence)**.** *Starting from any initial coalition structure Π_0, the proposed algorithm always converges to a final partition Π_f.*

Proof: The coalition formation phase can be mapped to a sequence of shift operations. That is, according to the hedonic shift rule, every shift operation transforms the current partition Π_c into another partition Π_{c+1}. Thus, starting from the initial step, the algorithms yields the following transformations:

$$\Pi_0 \rightarrow \Pi_1 \rightarrow \cdots \rightarrow \Pi_c \rightarrow \Pi_{c+1} \tag{5}$$

where the symbol $\rightarrow$ denotes the application of a shift operation. Every application of the shift rule generates two possible cases: (a) $\mathcal{S}_k \neq \emptyset$, so it leads to a new coalition partition, or (b) $\mathcal{S}_k = \emptyset$, so it yields a previously visited coalition partition with a non-cooperatively CP (i.e., with a coalition of size 1). In the first case, the number of transformations performed by the shift rule is finite (at most, it is equal to the number of partitions, that is the Bell number; see [25]), and hence the sequence in Eq. (5) will always terminate and converge to a final partition Π_f. In the second case, starting from the previously visited partition, at certain point in time, the non-cooperative CP must either join a new coalition and yield a new partition, or decide to remain non-cooperative. From this, it follows that the number of re-visited partitions will be limited, and thus, in all the cases the coalition formation stage of the algorithm will converge to a final partition Π_f. ■

Proposition 2 (Nash-stability)**.** *Any final partition Π_f resulting from the algorithm presented in Figure 1 is Nash-stable.*

Proof: We prove it by contradiction. Assume that the final partition Π_f is not Nash-stable. Consequently, there exists a CP $i \in \mathcal{N}$ and a coalition $\mathcal{S}_k \in \Pi_f \cup \emptyset$ such that $\mathcal{S}_k \cup \{i\} \succ_i \mathcal{S}_{\Pi_f}(i)$. Then, CP i will perform a hedonic shift operation and hence $\Pi_f \rightarrow \Pi'_f$. This contradicts the assumption that Π_f is the final outcome of our algorithm. ■

It is worth to point out that Nash-stability also implies the so called *individual-stability* [10]. A partition $\Pi = \{\mathcal{S}_1, \ldots, \mathcal{S}_l\}$ is *individually-stable* if do not exist a player $i \in \mathcal{N}$ and a coalition $\mathcal{S}_k \in \Pi \cup \emptyset$ such that $\mathcal{S}_k \cup \{i\} \succ_i \mathcal{S}_\Pi(i)$ and $\mathcal{S}_k \cup \{i\} \succeq_j \mathcal{S}_k$ for all $j \in \mathcal{S}_k$, i.e., if no player can benefit by moving from her/his coalition to another existing (possibly empty) coalition while not making the members of that coalition worse of. Thus, we can conclude that our algorithm always converges to a partition Π_f which is both Nash-stable and individually stable.

C. Computation of the Optimal Coalition Cost

To use the game-theoretic model discussed in the previous section, we need a way to find (for a given coalition) the optimal workload allocation (i.e., the allocation that minimizes the energy cost), that allows us to compute the coalition value.

To this end, we define a *Mixed Integer Linear Program* (MILP) modeling the problem of allocating a set $\mathcal{J}_\mathcal{S}$ of VMs onto a set $\mathcal{H}_\mathcal{S}$ of hosts so that the energy cost rate is minimized.

We base our MILP on the model described in [26], that has been first revised to improve its computational performance

$$\text{minimize } e = \sum_{i\in\mathcal{H}} \Big[p_i C^{\min}_{g(i)} + \alpha_i\big(C^{\max}_{g(i)} - C^{\min}_{g(i)}\big) + p_i(1-o(i))L_{g(i)} + (1-p_i)o(i)S_{g(i)}\Big]E_{c(i)} + \sum_{j\in\mathcal{J}} b_{ji} G_{c(h(j)),c(i),q(j)} \tag{6a}$$

subject to

$$\sum_{i\in\mathcal{H}} b_{ji} = 1, \qquad j \in \mathcal{J}, \tag{6b}$$

$$\sum_{j\in\mathcal{J}} b_{ji} \le |\mathcal{J}| p_i, \qquad i \in \mathcal{H}, \tag{6c}$$

$$\alpha_i \le p_i, \qquad i \in \mathcal{H}, \tag{6d}$$

$$\sum_{j\in\mathcal{J}} b_{ji} M_{q(j)g(i)} \le p_i, \qquad i \in \mathcal{H}, \tag{6e}$$

$$\sum_{j\in\mathcal{J}} b_{ji} A_{q(j)g(i)} = \alpha_i, \qquad i \in \mathcal{H}, \tag{6f}$$

$$b_{ji} \in \{0,1\}, \qquad j \in \mathcal{J}, i \in \mathcal{H}, \tag{6g}$$

$$\alpha_i \in \big[0,1\big], \qquad i \in \mathcal{H}, \tag{6h}$$

$$p_i \in \{0,1\}, \qquad i \in \mathcal{H}. \tag{6i}$$

Fig. 2. The VM allocation optimization model

TABLE I. EXPERIMENTAL EVALUATION – CONFIGURATION OF CPS

CP	# Hosts Class-1	Class-2	Class-3
CP_1	40	0	0
CP_2	0	40	0
CP_3	0	0	40
CP_4	15	15	10

TABLE II. EXPERIMENTAL EVALUATION – HOST CLASSES, VM CLASSES AND VM SHARES

(a) Characteristics of host classes

Host Class	CPU	RAM (GB)	$C^{\min}/C^{\max}$ (W)
1	2× Xeon 5130	16	86.7/274.9
2	Xeon X3220	32	143.0/518.4
3	2× Xeon 5160	64	490.1/1,117.8

(b) Characteristics of VM classes

VM Class	Processor	#CPUs	RAM (GB)
1	AMD Opteron 144	1	1
2	AMD Opteron 144	2	2
3	AMD Opteron 144	4	4

(c) Per-VM physical resource shares

Host Class	Class-1 VM	Class-2 VM	Class-3 VM
1	$(0.20, 0.062500)$	$(0.4, 0.12500)$	$(0.8, 0.2500)$
2	$(0.15, 0.031250)$	$(0.3, 0.06250)$	$(0.6, 0.1250)$
3	$(0.10, 0.015625)$	$(0.2, 0.03125)$	$(0.3, 0.0625)$

and then extended in order to incorporate the heterogeneity of physical resources and the energy cost.

The resulting optimization model is shown in Figure 2, where we use the same notation introduced in Section II,[3] and we denote with $o(i)$ the function that is 1 if host i is powered on and 0 otherwise (i.e., a function $o : \mathcal{H} \to \{0,1\}$), with L_k and S_k the power absorbed during the switch-on and switch-off operations of a host of class k, respectively, with $G_{i_1,i_2,v}$ the cost rate to migrate a VM of class v from CP i_1 to CP i_2, with E_i the cost rate of the energy consumed by a host belonging to CP i, with $c(i)$ the function that gives the CP that owns host i (i.e, a function $c : \mathcal{H} \to \mathcal{N}$), and with $h(j)$ the function that gives the host where VM j is allocated (i.e., a function $h : \mathcal{J} \to \mathcal{H}$).

In the optimization model we define, for any VM $j \in \mathcal{J}$ and host $i \in \mathcal{H}$, the following decision variables: b_{ji} is a binary variable that is equal to 1 if VM j is allocated to host i; α_i is a real variable representing the overall fraction of CPU assigned to all VMs allocated on host i; p_i is a binary variable that is equal to 1 if host i is powered on. The objective function $e\big(\mathcal{J},\mathcal{H}\big)$ (hereafter, e for short) represents, for a specific assignment of decision variables, the energy cost rate due to the power consumption induced by the federation of CPs to host the given VMs.

The resulting VM allocation is bound to the following constraints:

- Eq. (6b) imposes that each VM is hosted by exactly one host;
- Eq. (6c) states that only hosts that are switched on can have VMs allocated to them; the purpose of this constraint is to avoid that a VM is allocated to a host that will be powered off;
- Eq. (6d) ensures that (1) the CPU resource of a powered-on host is not exceeded, and (2) that no CPU resource is consumed on a host that will be powered off;
- Eq. (6e) assures that (1) the RAM resource of a powered-on host is not exceeded, (2) that VMs hosted on that host receive their required amount of RAM, and (3) that no RAM resource is consumed on a host that will be powered off;
- Eq. (6f) states that all VMs must exactly obtain the amount of CPU resource they require;
- Eq. (6g), Eq. (6h), and Eq. (6i) define the domain of decision variables b_{ji}, α_i, and p_i, respectively.

As in [26], in order to keep into considerations QoS requirements related to each class of VMs, we assumed that each VM j exactly obtains the amount of CPU $CPU_{q(j)}$ and RAM $RAM_{q(j)}$ as defined by its class $q(j)$ (see Section II).

IV. EXPERIMENTAL EVALUATION

To illustrate the effectiveness of our algorithm for coalition formation, we perform a set of experiments in which we compute the federation set for various scenarios including a population of distinct CPs.

In all scenarios, we consider $N = 4$ CPs, whose infrastructures are characterized as reported in Table I, and we use the host classes, VM classes and VM shares as the ones defined in Table II. We also assume that all CPs use the same revenue rate policy, that is they earn 0.08 \$/hour for class-1 VMs, 0.16

[3]To ease readability, we simplify it by denoting with $\mathcal{J}$ and $\mathcal{H}$ the cloud workload and the host set, respectively (i.e., we omit the dependence by $\mathcal{S}$).

\$/hour for class-2 VMs, and 0.32 \$/hour for class-3 VMs. Furthermore, without loss of generality, we also assume that the electricity price is the same for all CPs and it is equal to 0.4 \$/kWh.

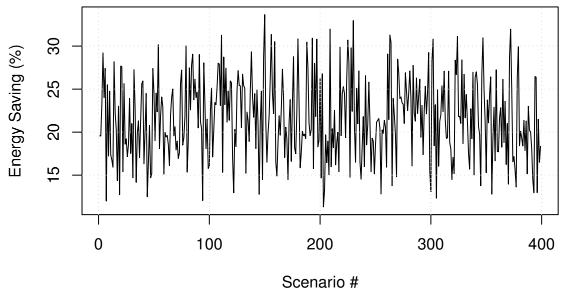
Starting from this configuration, we set up 400 scenarios that differ from each other in the workload of the various CPs, in the power state of each host and in the VM migration costs. Specifically, in each scenario the workload of each CP is set by randomly generating the number of VMs of each class as an integer number uniformly distributed in the $[0, 20]$ interval. In addition, to provide values to function $o(\cdot)$, we randomly generate the power state (i.e., ON or OFF) of each host according to a Bernoulli distribution with parameter 0.5. Furthermore, the values for L_k and S_k, for each host class k, are computed as the product of the electricity price, the maximum power consumption and the time taken to complete the switch-on or switch-off operation. This switch-on/-off time is randomly generated for each host class according to a Normal distribution with mean of 300 μsec and standard deviation (S.D.) of 50 μsec (e.g., see [27]). Finally, the VM migration costs $G_{c_1,c_2,k}$ from CP c_1 to CP c_2 for each VM class k are computed as the product between the data transfer cost rate, the data size to transfer and the time to migrate a VM of class k from CP c_1 to CP c_2, and assuming that our algorithm activates every 12 hours. The data transfer cost rate is taken from the Amazon EC2 data transfer pricing [13] and set to 0.001 \$/GB. Furthermore, we suppose that data are persistently transferred during the migration time at a fixed data rate of 100 Mbit/sec for all CPs. For what concerns the migration time, we assume that it is randomly generated according to a Normal distribution with mean of 277 sec and S.D. of 182 sec for VMs of class 1, with mean of 554 sec and S.D of 364 sec for VMs of class 2, and with mean of 1108 sec and S.D. of 728 sec for VMs of class 3 (e.g., see [28]). The migration cost between hosts of the same CP is assumed to be negligible.

For each one the above scenario, we compute the federation set of the involved CPs by using an ad-hoc simulator written in C++ and interfaced with CPLEX [29] to solve the various instances of the optimization model of Section III-C.

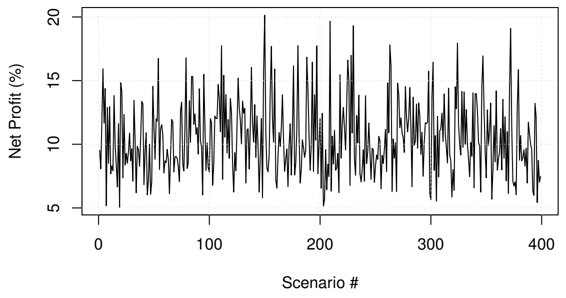
In the rest of this section, we first present a summary of the performance obtained by our algorithm over all scenarios, and then, we illustrate its behavior by showing its run trace for one of these scenarios.

In Figure 3, we compare the performance of each scenario in terms of energy saving and net profit obtained with our algorithm with respect to the *no-federation* case (i.e., when CPs work in isolation). Specifically, the figures show, for each scenario, the percentage of the reduction of energy consumption (see Figure 3a) and of the increment of net profit (see Figure 3b) that all CPs obtain when they federate according to our algorithm with respect to the case of working individually. As can be seen from these figures, *our algorithm results in an reduction of the overall consumed energy from* 11.3% *to* 33.6% *(with an average of* 21.6%*), and in an increment of the overall net profit from* 5.1% *to* 20.1% *(with an average of* 10.5%*)* with respect to the case where CPs work non-cooperatively.

We can also analyze the benefits provided by our algorithm from the point of view of each CP. Results from our experiments show that, from the CP perspective, the formation of federations yielded by our algorithm is always non-detrimental. Specifically, it results that, on average, the net profit earned by CP$_1$, CP$_2$, CP$_3$ and CP$_4$ increases by nearly 18.0%, 8.5%, 22.8% and 4.3% with respect to the no-federation case, respectively.



(a) Reduction of energy consumption (%)



(b) Increase of net profit (%)

Fig. 3. Performance of our algorithm with respect to the no-federation case

TABLE III. EXPERIMENTAL RESULTS – WORKLOAD OF CPS IN THE CASE STUDY

CP	# VMs Class-1	# VMs Class-2	# VMs Class-3
CP$_1$	0	12	13
CP$_2$	18	5	11
CP$_3$	17	18	11
CP$_4$	3	2	0

Finally, to illustrate how our algorithm works, we present the run trace for a single scenario, whose characteristics are reported in Table III. [4] We select this scenario to illustrate the behavior of the algorithm when there are multiple Nash-stable partitions. [5]

For this investigation, we show in Table IV all possible partitions together with the value function $v(\cdot)$ of every coalition inside each partition, and the corresponding Shapley values. [6] From the table, we can see that there are two Nash-stable coalitions, namely $\{1, 2, 3, 4\}$ and $\{\{1, 3\}, \{2, 4\}\}$. To get one of these partitions, our algorithm works as follows. Starting from partition $\{\{1\}, \{2\}, \{3\}, \{4\}\}$ (i.e., every CP works individually), there are two different sequences of hedonic shift rules:

- Sequence #1: $\{\{1\}, \{2\}, \{3\}, \{4\}\} \xrightarrow{3}$

[4]Due to lack of space, we only report the number of VMs.

[5]Note, our algorithm's output is always a single partition.

[6]Note, our algorithm does not necessarily enumerate all of such partitions.

$\{\{1,3\},\{2\},\{4\}\} \xrightarrow{2} \{\{1,2,3\},\{4\}\} \xrightarrow{4} \{\{1,2,3,4\}\}$

- Sequence #2: $\{\{1\},\{2\},\{3\},\{4\}\} \xrightarrow{3} \{\{1,3\},\{2\},\{4\}\} \xrightarrow{2} \{\{1,3\},\{2,4\}\}$

where the index on top of each arrow denotes the CP that performs the corresponding hedonic shift rule.

Regardless what partition is finally selected, from the third column of Table IV we can also observe that, for this scenario, the partition value improvement for both Nash-stable partitions with respect to the non-cooperative behavior (i.e., partition $\{\{1\},\{2\},\{3\},\{4\}\}$) is about 16.5% for partition $\{\{1,3\},\{2,4\}\}$ and nearly 17.2% for the grand-coalition.

V. Related Works

Recently, the concept of cloud federations [6], [30] has been proposed as a way to provide individual CPs with more flexibility when allocating on-demand workloads. Existing work on cloud federations has been mainly focused on the development of architectural models for federations [31], and of mechanisms providing specific functionalities (e.g., workload management [32], [33], accounting and billing [34], and pricing [35]–[38]).

To the best of our knowledge, very little work has been carried out to jointly tackle the problem of dynamically forming stable cloud federations for energy-aware resource provisioning. Indeed, much of the existing work only focuses on a single aspect of the problem. In [39], the design and implementation of a VM scheduler for a federation of CPs is presented. The scheduler, in addition to manage resources that are local to each CP, is able to decide when to rent resources from other CPs, when to lease own idle resources to other CPs, and when to turn on or off local physical resources. Unlike our work, this work does not consider the problem of forming stable CP federations. In [40], a cooperative game-theoretic model for federation formation and VM management is proposed. In this work, the federation formation among CPs is analyzed using the concept of network games, but the energy minimization problem is not considered.

In [11], a profit-maximizing game-based mechanism to enable dynamic cloud federation formation is proposed. The dynamic federation formation problem is modeled as a hedonic game (like our approach), and the federations are computed by means of a merge-split algorithm. There are several important differences between this and our works: (1) we focus on the stability of individuals rather than of groups, (2) we propose a decentralized algorithm, (3) we demonstrate the stability of the obtained federations, and (4) we use the Shapley value instead of the normalized Banzhaf value (as in [11]), since the latter does not satisfy some important properties [41].

In [42], the problem of sharing unused capacity in a federation of CPs for VM spot market is formulated as a non-cooperative repeated game. Specifically, by using a Markov model to predict future non-spot workload, the authors introduce a set of capacity sharing strategies that maximize the federation's long-term revenue and propose a dynamic programming algorithm to find the allocation rules needed to achieve it. Our work can complement this approach by providing a solution to the formation of CP federations for non-spot VM instances.

VI. Conclusions and Future Works

This paper investigates a novel dynamic federation scheme among a set of CPs. To this end, we propose a cooperative game-theoretic framework to study the federation formation problem, and a mathematical optimization model to allocate CP workload in an energy-aware fashion, in order to reduce CP energy costs.

Unlike similar proposals, we adopt a distributed approach in which each CP autonomously and selfishly makes its own decisions, and the best solution emerges from these decisions without the need to synchronize them or to resort to a trusted third party, and such that no CP can benefit by moving from her/his federation to another (possibly empty) one.

In the proposed scheme, we model the cooperation among the CPs as a coalition game with transferable utility and we devise a distributed hedonic shift algorithm for coalition formation. With the proposed algorithm, each CP individually decides whether to leave the current coalition to join a different one according to his preference, meanwhile improving the perceived net profit. Furthermore, we prove that the proposed algorithm converges to a Nash-stable partition which determines the resulting coalition structure. Numerical results show the effectiveness of our approach.

The future developments of this research is following several directions. First of all, we would like to enhance the coalition value function in order to account for possible request losses due to lack of physical resources. Furthermore, we want to improve the game-theoretic and optimization models in order to include costs in terms of loss of revenues as well as other aspects like the ones related to trustworthiness among CPs.

As a second direction, we plan to integrate the long-term resource provisioning solution proposed in this paper with other short-term and medium-term resource management strategies (e.g., [5], [43]) to improve resource utilization and meet application-level performance requirements, and with techniques for incremental VM migration (e.g., [44]).

Finally, we want to implement and validate the proposed algorithm in a real testbed.

References

[1] P. Barham et al., "Xen and the art of virtualization," in *Proc. of the* 19*th SOSP*, 2003.

[2] "VMware Inc." http://www.vmware.com.

[3] T. E. S. Program, "Report to Congress on server and Data Center energy efficiency," U.S. EPA, Tech. Rep., Aug 2007.

[4] M. Guazzone et al., "Energy-Efficient Resource Management for Cloud Computing Infrastructures," in *Proc. of the* 3*rd CloudCom*, 2011.

[5] L. Albano et al., "Fuzzy-Q&E: achieving QoS guarantees and energy savings for cloud applications with fuzzy control," in *Proc. of the* 3*rd CGC*, 2013.

[6] B. Rochwerger et al., "RESERVOIR – When one cloud is not enough," *Computer*, vol. 44, no. 2, 2011.

[7] A. Celesti et al., "Towards energy sustainability in federated and interoperable clouds," in *Communication Infrastructures for Cloud Computing*, H. Mouftah et al., Ed., 2013.

TABLE IV. EXPERIMENTAL RESULTS – COALITION VALUES AND SHAPLEY VALUES FOR ALL THE 15 DIFFERENT PARTITIONS OF THE CASE STUDY

$\Pi = \{\mathcal{S}_1, \ldots, \mathcal{S}_l\}$	$\big\{v(\mathcal{S}_1), \ldots, v(\mathcal{S}_l)\big\}$	$\sum_{\mathcal{S}_i \in \Pi} v(\mathcal{S}_i)$	$\{\phi_{\mathcal{S}_1}, \ldots, \phi_{\mathcal{S}_l}\}$
$\big\{\{1\},\{2\},\{3\},\{4\}\big\}$	$\{4.28, 3.45, 3.84, 0.38\}$	11.95	$\big\{\{4.28\},\{3.45\},\{3.84\},\{0.38\}\big\}$
$\big\{\{1,2\},\{3\},\{4\}\big\}$	$\{8.22, 3.84, 0.38\}$	12.44	$\big\{\{4.52, 3.70\},\{3.84\},\{0.38\}\big\}$
$\big\{\{1,3\},\{2\},\{4\}\big\}$	$\{9.59, 3.45, 0.38\}$	13.42	$\big\{\{5.01, 4.57\},\{3.45\},\{0.38\}\big\}$
$\big\{\{1\},\{2,3\},\{4\}\big\}$	$\{4.28, 7.82, 0.38\}$	12.48	$\big\{\{4.28\},\{3.72, 4.10\},\{0.38\}\big\}$
$\big\{\{1,4\},\{2\},\{3\}\big\}$	$\{4.69, 3.45, 3.84\}$	11.98	$\big\{\{4.29, 0.40\},\{3.45\},\{3.84\}\big\}$
$\big\{\{1\},\{2,4\},\{3\}\big\}$	$\{4.28, 4.33, 3.84\}$	12.45	$\big\{\{4.28\},\{3.70, 0.63\},\{3.84\}\big\}$
$\big\{\{1\},\{2\},\{3,4\}\big\}$	$\{4.28, 3.45, 5.43\}$	13.16	$\big\{\{4.28\},\{3.45\},\{4.44, 0.99\}\big\}$
$\big\{\{1,2,3\},\{4\}\big\}$	$\{13.27, 0.38\}$	13.65	$\big\{\{5.00, 3.70, 4.57\},\{0.38\}\big\}$
$\big\{\{1,2,4\},\{3\}\big\}$	$\{8.69, 3.84\}$	12.53	$\big\{\{4.39, 3.80, 0.50\},\{3.84\}\big\}$
$\big\{\{1,2\},\{3,4\}\big\}$	$\{8.22, 5.43\}$	13.65	$\big\{\{4.52, 3.70\},\{4.44, 0.99\}\big\}$
$\big\{\{1,3,4\},\{2\}\big\}$	$\{10.01, 3.45\}$	13.46	$\big\{\{4.63, 4.78, 0.60\},\{3.45\}\big\}$
$\big\{\{1,3\},\{2,4\}\big\}$	$\{9.59, 4.33\}$	13.92	$\big\{\{5.01, 4.57\},\{3.70, 0.63\}\big\}$
$\big\{\{1,4\},\{2,3\}\big\}$	$\{4.69, 7.82\}$	12.51	$\big\{\{4.29, 0.40\},\{3.72, 4.10\}\big\}$
$\big\{\{1\},\{2,3,4\}\big\}$	$\{4.28, 8.88\}$	13.16	$\big\{\{4.28\},\{3.62, 4.36, 0.90\}\big\}$
$\big\{1,2,3,4\big\}$	$\{14.01\}$	14.01	$\big\{4.78, 3.78, 4.76, 0.68\big\}$

[8] B. Peleg et al., *Introduction to the Theory of Cooperative Games*, 2nd ed. Springer Berlin Heidelberg, 2007.

[9] D. Ray, *A Game-Theoretic Perspective on Coalition Formation*, ser. The Lipsey Lectures. Oxford University Press, 2007.

[10] A. Bogomolnaia et al., "The Stability of Hedonic Coalition Structures," *Game Econ Behav*, vol. 38, pp. 201–230, 2002.

[11] L. Mashayekhy et al., "A coalitional game-based mechanism for forming cloud federations," in *Proc. of* 5*th UCC*, 2012.

[12] S. Rivoire et al., "A comparison of high-level full-system power models," in *Proc. of the HotPower*, 2008.

[13] "Amazon Elastic Compute Cloud," http://aws.amazon.com/ec2.

[14] "GeekBench: Next-generation processor benchmark," http://www.primatelabs.com/geekbench.

[15] J. Drèze et al., "Hedonic coalitions: Optimality and stability," *Econometrica*, vol. 48, no. 4, pp. 987–1003, 1980.

[16] K. Apt et al., "A Generic Approach to Coalition Formation," *Int Game Theor Rev*, vol. 11, no. 03, pp. 347–367, 2009.

[17] L. S. Shapley, "A Value for n-person Games," in *Contributions to the Theory of Games*, H. Kuhn et al., Ed., 1953, pp. 307–317.

[18] R. Aumann et al., "Cooperative games with coalition structures," *Int J Game Theor*, vol. 3, no. 4, pp. 217–237, 1974.

[19] W. Saad et al., "Hedonic coalition formation for distributed task allocation among wireless agents," *IEEE Trans Mobile Comput*, vol. 10, no. 9, pp. 1327–1344, 2011.

[20] ——, "Coalition formation games for distributed cooperation among roadside units in vehicular networks," *IEEE J Sel Area Comm*, vol. 29, no. 1, pp. 48–60, 2011.

[21] ——, "A selfish approach to coalition formation among unmanned air vehicles in wireless networks," in *Proc. of the* 1*st GameNets*, 2009.

[22] G. Coulouris et al., *Distributed Systems: Concepts and Design*, 5th ed. Addison Wesley, 2011.

[23] G. Weiss, Ed., *Multiagent Systems*, 2nd ed. MIT Press, 2013.

[24] A. Kshemkalyani and M. Singhal, *Distributed Computing: Principles, Algorithms, and Systems*. Cambridge University Press, 2008.

[25] G. Rota, "The number of partitions of a set," *Am Math Mon*, vol. 71, no. 5, pp. 498–504, 1964.

[26] D. Borgetto et al., "Energy-aware service allocation," *Future Generat Comput Syst*, vol. 8, no. 25, pp. 769–779, 2012.

[27] D. Meisner et al., "PowerNap: Eliminating server idle power," in *Proc. of the* 14*th ASPLOS*, 2009.

[28] S. Akoush et al., "Predicting the performance of virtual machine migration," in *Proc. of the MASCOTS*, 2010.

[29] "IBM ILOG CPLEX Optimizer," http://www.ibm.com/software/integration/optimization/cplex-optimizer.

[30] R. Moreno-Vozmediano et al., "IaaS Cloud Architecture: From Virtualized Data Centers to Federated Cloud Infrastructure," *Computer*, vol. 45, no. 12, pp. 65–72, 2012.

[31] A. Ferrer et al., "OPTIMIS: A Holistic Approach to Cloud Service Provisioning," *Future Generat Comput Syst*, vol. 28, no. 1, pp. 66–77, 2012.

[32] A. Nordal et al., "Balava: Federating Private and Public Clouds," in *Proc. of the* 7*th SERVICES*, 2011.

[33] L. Larsson et al., "Scheduling and Monitoring of Internally Structured Services in Cloud Federations," in *Proc. of the* 16*th ISCC*, 2011.

[34] E. Elmroth et al., "Accounting and Billing for Federated Cloud Management," in *Proc. of the* 8*th GCC*, 2009.

[35] M. Hassan et al., "Cooperative game-based distributed resource allocation in horizontal dynamic cloud federation platform," *Inform Syst Front*, pp. 1–20, 2012.

[36] J. Künsemöller et al., "A game-theoretical approach to the benefits of cloud computing," in *Proc. of the* 8*th GECON*, 2012.

[37] M. Mihailescu et al., "Dynamic Resource Pricing on Federated Clouds," in *Proc. of the* 10*th CCGrid*, 2010.

[38] A. Toosi et al., "Resource provisioning policies to increase iaas provider's profit in a federated cloud environment," in *Proc. of the* 13*th HPCC*, 2011.

[39] Í. Goiri et al., "Economic model of a cloud provider operating in a federated cloud," *Inform Syst Front*, vol. 14, no. 4, pp. 1–17, 2012.

[40] D. Niyato et al., "Cooperative virtual machine management for multi-organization cloud computing environment," in *Proc. of the* 5*th VALUETOOLS*, 2011.

[41] R. van den Brink et al., "Axiomatizations of the normalized Banzhaf value and the Shapley value," *Soc Choice Welfare*, vol. 15, no. 4, pp. 567–582, 1998.

[42] N. Samaan, "A novel economic sharing model in a federation of selfish cloud providers," *IEEE Trans Parallel Distr Syst*, vol. 25, no. 1, pp. 12–21, 2014.

[43] M. Guazzone et al., "Exploiting VM migration for the automated power and performance management of green cloud computing systems," in *Proc. of the* 1*st E2DC*, 2012.

[44] A. Verma et al., "pMapper: Power and migration cost aware application placement in virtualized systems," in *Proc. of the* 9*th Middleware*, 2008.